Experiences of Modelling and Using Object-Oriented Telecommunication Service Frameworks in SDL

Mario Kolberg

Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow G1 1XW, United Kingdom, mkolberg@acm.org

Richard O. Sinnott

GMD-Fokus, Kaiserin-Augusta-Allee 31, Berlin, Germany, sinnott@fokus.gmd.de

Evan H. Magill

Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow G1 1XW, United Kingdom,
e.magill@eee.strath.ac.uk

Abstract

This paper describes experiences in using SDL and its associated tools to create telecommunication services by producing and specialising object-oriented frameworks. The chosen approach recognises the need for the rapid creation of validated telecommunication services. It introduces two stages to service creation; firstly a software expert produces a service framework, and secondly a telecommunications 'business consultant' who specialises the framework by means of graphical tools to rapidly produce services. Here we focus on the underlying technology required. In particular we highlight the advantages of SDL and tools as well as issues and problems incurred.

Key words: Object-Oriented Frameworks, Service Creation, SDL, TINA, Testing, TTCN.

1 Introduction

The area of Service Creation, while far from new, grows ever more complex and challenging due to the constant increase in requirements placed on it by

consumer demands, competition and new technologies. Many new factors need to be considered to ensure that the general requirements on service creation are met. Some of these requirements include:

- Reduction in the time to market for both new services and variants on existing services,
- Reductions in development and operating costs,
- The need for open solutions to service creation, i.e. methods and processes which are equally applicable in many software environments,
- The facility to provide prototype services, for the purpose of quality assurance and validation of user requirements,
- Re-use of existing software/components,
- Speed of interaction and correct interworking of new services with existing (or legacy) infrostructure.

Classic software engineering approaches are insufficient to meet these challenges. Therefore, new and innovative service creation processes are required. This paper reports on work carried out in the EU funded ACTS TOSCA (TINA Open Service Creation Architecture) project which is addressing these issues by developing a methodology and a suite of tools for the rapid creation and validation of TINA-like services. This methodology is based on the combination of frameworks and paradigm tools. While object-oriented frameworks are not new to the software engineering community [1–8], their application in the service creation process of telecommunication services is believed to be novel. Furthermore, and this is a vital point in the TOSCA approach, the use of paradigm tools in combination with frameworks allows for widening the participation in the service creation process. Paradigm tools offer a graphical and intuitive means whereby services can be designed. In other words, paradigm tools abstract from the complexity of the frameworks and offer a view on their functionality which is accessible by non-technical people, e.g. business consultants. Thus the service designer does not need to consider the lower level behaviour of the service to be able to create one. Rather, they should be provided with a high-level representation of the service components and the ability to tune their behaviour and how they are composed with one another. In other words, the creation of services is moved to a large extent from the technical labs to the front offices where business consultants deal with potential customers. Hence, almost immediate feedback on the behaviour of the new services can be delivered and potential changes demanded by the customer can be taken into account. More information on paradigm tools and the paradigm based service creation process more generally are given in [9–11].

The aims of the TOSCA project are to develop a service creation environment that enables multimedia-based telecommunication services to be produced in an effective manner, i.e. they are created rapidly but not at the expense of their reliability or quality [9,12]. Central to the approach is that the services

to be generated are validated. This validation is required both when the service is initially created and also when it is deployed in an environment where it may interwork with other services causing potentially undesired service interactions [13,14].

Validation of services implies that formality is introduced into the service creation process. Producing formal specifications of the system to be developed is a traditional starting point in applying formal techniques [15]. Unfortunately, it is often the case that formal techniques are used only at this stage of the software development process. Ideally, formality should be taken through to the final implementation of the software itself. This is a notoriously difficult activity – often depending upon the nature of the formal language and method – requiring arduous refinement and obligatory proof steps. An alternative process to refinement of specifications through to their final implementation is to develop the specification and implementation as dual, i.e. concurrent, activities. Provided that the specification and implementation are at the same level of abstraction, the specification can be used as a basis for testing the implementation.

Distributed system development offers one area where the parallels between the development of specification and implementation can be readily drawn, i.e. they can be expressed at the same levels of abstraction. Interface definition languages (IDL) when used as a common vocabulary for describing the syntactic aspects of interface interactions, serve as an ideal starting point for developing both specifications and implementations [16].

Given that the rapid development of high quality services is a fundamental aspect of service creation in TOSCA, developing specifications (and implementations) from nothing, or from an IDL only basis, is not a viable option. Instead techniques that can expedite the software development process are necessary. Whilst it is typically the case that implementations rarely (if ever!) start from nothing, the same cannot be said for the development of formal specifications [17,18]. In TOSCA we are addressing this issue through the adoption of techniques based upon object-oriented frameworks.

The concept of framework based software engineering has arisen to help to realise the holy grail of software engineering: re-use. Frameworks are a natural extension of object-oriented techniques. Whilst object technology [19] provides a basis for re-use of code, it does not provide features to capture the design experience as such. Frameworks have developed to fulfil this need. A framework can be regarded as a collection of pieces of software or specification fragments that have been developed to produce software of a certain type or niche [20]. A framework is only partially complete. Typically, they are developed so that they have holes or flexibility points in them where service specific information is to be inserted. This filling in (specialisation) of the flexibility points is used

to develop a multitude of services with differing characteristics.

Following the approach of parallel development of the specification and implementation, in TOSCA the frameworks are developed both in the implementation world, using C++ and distributed technologies such as CORBA [16] and the specification world, using SDL [21]. SDL is frequently used in creating telecommunications services. Used with suitable tools, SDL can provide support for service development from requirements capturing to testing.

As outlined above, the TOSCA approach uses paradigm tools to facilitate the specialisation of the frameworks by business consultants. Our focus in this paper, however, is on the development and usage of SDL frameworks. Specifically, we identify the advantages and disadvantages of applying SDL and it's associated tools throughout the TOSCA based service creation lifecycle.

2 The TOSCA approach to Service Creation

The TOSCA project is developing a service creation environment where services can be created and validated in an expedited manner. Tool support forms a central part of the TOSCA approach. TOSCA has developed a tool chain that allows for both the development and usage of specification frameworks from semi-formal descriptions right through to their usage in testing the created service. Figure 1 highlights this tool chain.

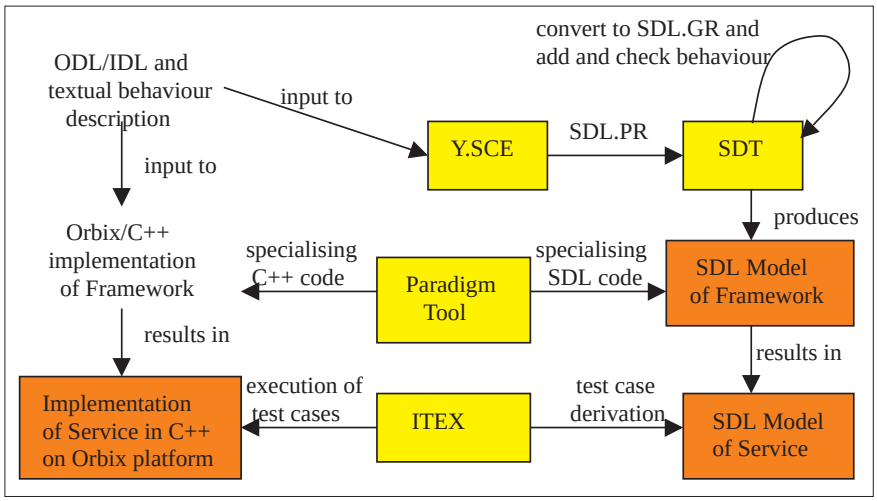

Fig. 1. The Tool Chain in TOSCA

Here the Y.SCE tool [22] allows (amongst other things) TINA ODL [23] and CORBA IDL [16] descriptions to be developed (or imported) and subsequently mapped to SDL in PR format. These SDL fragments are then themselves imported into the Telelogic TAU toolset [24]. This toolset consists of a collection

of tools that allow SDL specifications to be both specified, simulated or validated (using the Specification Design Tool (SDT) tool) and subsequently used for generating test cases (using the Interactive TTCN Editor and eXecutor (ITEX) tool). The requirements on the behaviour of the framework are represented both by use cases and textual descriptions of the expected behaviour of the framework components. Simulation techniques are used to ensure that the framework has the correct behaviour, e.g. that it satisfied the use cases.

When complete the SDL framework model can be used to generate test cases. These can then be used to test both that the SDL service models are valid, i.e. services created from the framework, as well as a minimum conformance requirement on C++/CORBA based service implementations.

The SDL model of the framework is then saved as a package which can then be used by paradigm tools to develop complete models of services. TOSCA has implemented two paradigm tools that can be used to produce intuitive (graphical) models of the services. We consider one in particular based on the functional block paradigm. This paradigm provides service designers with a list of basic events at which the behaviour of the service can be defined. These are the key points at which the designer can intervene and customise the service behaviour. The basic events thus correspond to the framework flexibility points. Numerous basic events have been identified, e.g. starting/stopping the service, starting/stopping user sessions, etc. We focus on the form of these flexibility points and the behaviour that can be inserted into them in sections 5 and 6.

Once the user of the paradigm tool is satisfied with the design of the service, the paradigm tool outputs both the specialising C++ and SDL. The generated SDL is then imported into SDT and used to develop an SDL system based on the framework package. Once complete, the SDL service specification is checked for minimum conformance through ensuring it passes all test cases contained in the framework test suite. When this is the case, the SDL service specification itself is used to generate test cases for the C++ based service implementation. These test cases may be executed against the C++/CORBA based service implementations through a TTCN/CORBA gateway. Information on how CORBA based systems can be tested and the gateway itself are provided in [25,26].

The first stage in this tool chain is the development of the framework description. This is represented through TINA ODL and CORBA IDL descriptions with associated use cases and textual descriptions of the object and interface behaviours. The actual framework itself is based around the TINA architecture, or more specifically the Service Architecture [27] of TINA.

3 Frameworks based on the TINA Architecture

The TINA Service Architecture introduces the underlying concepts and provides information on how telecommunication applications and the components they are built from, have to behave. Central to the Service Architecture is the concept of a session. This is defined as a temporary relationship between a group of resources assigned to collectively fulfil a task or objective.

Three sessions are defined in TINA: the Access Session, Service Session and Communication Session. Briefly, the access session provides mechanisms to support access to services (service sessions) that have been subscribed to. The service session allows users to execute and manage sessions, i.e. it allows control of the communication session. The communication session controls the network resources required to establish end to end connections.

Currently, the service session has been the main area upon which frameworks are being developed in TOSCA. The components in the service session and the relation between the three sessions are depicted in Figure 2.

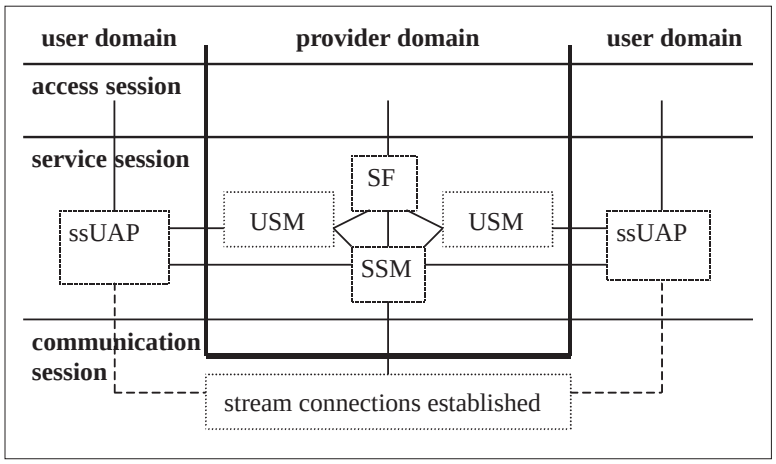

Fig. 2. Relation between the TINA Sessions

Broadly speaking, an instance of a service typically consists of a Service Session Manager (SSM) to control the global service behaviour, and a collection of User Service Session Managers (USM) - one of each is used to control a users participation in that service. Both types of components are instantiated by the Service Factory (SF) when requested to do so by components of the Access Session. The Service Session related User Application (ssUAP) represents a set of applications in the user domain which allow a user to communicate with a service.

The USM and SSM components in the framework are decomposed into generic and specific parts with the generic parts being fixed and the specific parts being incomplete in the framework and thus specialiseable by the paradigm

tool. Figure 3 gives an overview of the USM component structure and its relation to a typical ssUAP and SSM.

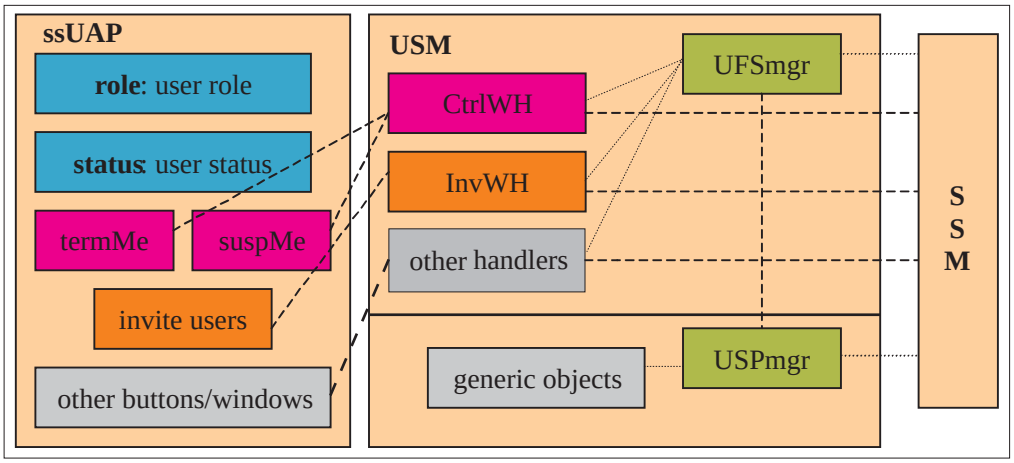

Fig. 3. Structure of the USM and Relation to ssUAP and SSM

Each of the service parts detailed above has a manager (UFSmgr, USPmgr for the USM and GFSmgr, GSPmgr for the SSM not shown here). These managers are responsible for lifecycle and initial access to the managed objects, e.g. the managers are able to initialise, suspend, resume or terminate the objects they manage, or provide references to the objects they manage on request. When the manager is told to suspend, resume or terminate itself it also suspends, resumes or terminates the objects it manages respectively. As we shall see in section 5, these manager operations and the initialisation of the manager together with the objects it is to create (and subsequently manage) correspond to framework flexibility points.

Typically, users can join, suspend, resume or terminate their participation in services. The logic associated with these requests are processed in the service session, e.g. whether the user is able to resume themselves in the service at that time. If successful, the appropriate operations are invoked on the communication session, e.g. resume my previously suspended connections.

It is important to note that this architecture does not overly constrain the kinds of services that can be created from it. Rather, it acts as a template for a multitude of services, e.g. multimedia conferencing services, chatline services or newsflash services to name but three. Indeed even within these three services there exist a plethora of variations. In multimedia conferencing for example, there might be differing roles, e.g. chairman, observer, participant. These differing roles might result in differing expected functionalities, e.g. only chairman can invite (or suspend or terminate) other users, only participants can vote. Users might be able to have differing charging (or billing or accounting) possibilities, e.g. reverse or split charging, or other variations.

As well as these role specific specialisations, numerous others are possible also,

e.g. only start the service if a certain number of successful responses to the invite have been received. Quit the service session if the number of users falls below a certain level (or if the total charges generated from using the service falls below a certain level). Terminate a user if they have been suspended for too long. It is precisely these variations on the general theme that paradigm tools are expected to capture whilst the general theme itself is represented by the framework.

To engineer frameworks it is thus necessary to have a core behaviour. In TOSCA this core behaviour is based around the informal (textual) description of the behaviour of the service session components, along with the TINA ODL and IDL specification for those objects. TINA ODL is a superset of IDL which allows, amongst other things, to specify objects which offer multiple interfaces to their environment. Further, TINA ODL distinguishes between supported and required interfaces. ODL also allows for the expression of groups of objects and the objects used to manage those groups.

4 Tool supported Mapping from ODL to SDL

Given that TINA component specifications are written in TINA ODL [23], supporting a TINA ODL mapping is critical if the framework based approach is to be successful. Table 1 summarises the main rules of the ODL to SDL mapping used in TOSCA and supported by the Y.SCE tool [28].

4.1 Advantages of the Mapping

The greatest advantage of the mapping used is that it offers a basis for comparison of the SDL model and the C++/CORBA based implementation. Often detractors of formal methods cite that formal models of systems bear little or no relation to the actual software development itself. This is often a deliberate policy, e.g. where a requirements specification is made. Having a common (syntactic) basis for the intercommunication between the objects in the SDL world and in the C++/CORBA implementation worlds addresses this issue directly. Put another way, the formal model and software implementation can be developed at the same level of abstraction. Through this, the model can be used directly by tools etc. for reasoning about and testing the implementation or parts of the implementation.

One major advantage of this mapping to others [29] is that it allows for exceptions in the SDL model. Exceptions are an essential feature in distributed systems, moreover, ODL also supports exceptions. The support for exceptions

Table 1
Summary of the ODL to SDL Mapping

ODL/IDL Structure	**SDL Mapping**
group type	block type
object type	block type
interface type	process type
object reference	Pid
oneway (asynchronous) operation)	signal prefixed with pCALL_
operation (synchronous)	signal pair. The first signal is prefixed with pCALL_, the second signal is prefixed with pREPLY_ or pRAISE_ (if exception raised).
exception	signal prefixed with pRAISE_
basic IDL types, e.g. long, char, float, ...	syntype
any	not supported
enum	newtype with corresponding literals
typedef	syntype
struct	newtype with corresponding structure
constant	sysnonym

is gained through mapping ODL operations to SDL signals pairs, as opposed to remote procedures. Remote procedures are a shorthand notation and use a substitution model based on signal pairs and states. More precisely, remote procedures are decomposed into two signals. The first carries the outgoing parameters (in or inout) and the second the return value of the procedure and all inout parameters. These signals are sent via implicit channels and signalroutes. As these signals are only internally generated and thus not visible within a specification, it is impossible to return with a different signal such as an exception signal.

As with other ODL language mappings, the Y.SCE tool generates client stubs and server skeletons. The generated SDL is placed into packages, which are ready to use in subsequent specification steps. In TOSCA these packages were converted to SDL.GR format and imported into the SDT tool.

Whilst overcoming certain problems with other mappings, e.g. lack of exceptions, the Y.SCE mappings are also not without problems. One problem with this and other IDL mappings is that it reduces the advantages that can be gained from the technique of abstraction. Working at the IDL and ODL level when modelling a realistic system, e.g. a telecommunication service, means that it is more difficult to get the big-picture of what the system is doing. This is lost to a certain extent through the often low level interactions of the objects in the system.

There are further side effects of this abstraction problem that become apparent when tools are used to check the SDL system, e.g. when trying to validate the system through performing state space exploration. Having several hundred objects interacting in non-trivial ways carrying complex parameters detracts from the ability of the tools to work successfully.

As well as the abstraction problem, there are other differences and associated problems with the mappings when interpreted from a CORBA perspective. The current CORBA specification does not directly support objects having multiple interfaces as do other architectures, e.g. the Open Distributed Processing Reference Model [30]. Multiple interface objects are currently under investigation and may well be in the next CORBA 3.0 specification. Having block types with multiple process types, i.e. objects types with multiple interface types, requires design differences to be made between the SDL framework and C++ frameworks. For example, CORBA objects have behaviour but SDL blocks do not. Block behaviour in SDL is only given through the processes a block contains. Similarly, CORBA objects can have data associated with them. Data may not be declared at the block level in SDL. Instead processes must be specified to either reveal data or export data structures which can be viewed or imported by the processes within that block respectively. Alternatively, additional signals can be added between the processes to access the relevant data. This is not an ideal solution however since it increases the communications necessary between the processes and can result in poorer run-time performance.

A further issue connected to the ODL/SDL mapping is related to the threading models used in C++ and SDL. The chosen threading model for the C++/CORBA implementation is such that there is only one single thread per ODL object group. However, in the SDL model this can not be achieved in a straightforward way. In fact, since all ODL interfaces are mapped to process types, instantiations of these can accept requests from other processes concurrently. In other words, since the processing of the request is done in the interface as opposed to the object, multiple requests to the same object at

different interfaces, can be processed in parallel. Although also this issue can be solved by added communication it is somewhat inellegant.

Another related difference between the current CORBA specification and the SDL mapping is that CORBA object references are first class citizens, i.e. they may be passed around as parameters in operations. This facility enables dynamic systems to be built where new resources can be found and subsequently bound to at run time. Blocks are not first class entities in SDL, e.g. they cannot be passed around as parameters in signals. To overcome this, processes representing the core block behaviour can be specified. These processes typically manage the other processes (representing the object interfaces) in the block. References to these manager processes can subsequently be passed around as parameters in signals.

Another discrepancy between the CORBA and SDL worlds is the dynamic creation of objects. As stated, USMs and SSMs are dynamically created by the service factory on request from users (and they themselves can dynamically create objects when requested). Since objects are represented as block types, instances of these cannot be created dynamically and an alternative solution is required.

One possibility is to have process instances inside blocks that exist at system start-up. For example, at start-up, the UFS block contains a creator process used only to create instances of the manager process type inside the UFS. This manager process can then create instances of other process types as required. The reference (PId) to this manager is returned to the invoker as illustrated in Figure 4.

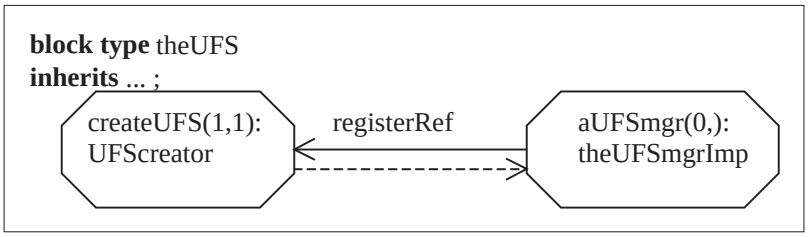

Fig. 4. Overcoming the lack of dynamic block creation in SDL

Typically, these creator processes (UFScreator) support a single exported procedure which is imported into blocks wishing to create "instances" of the exporting block. For example, the service factory will import and can subsequently call the exported remote procedure for the UFSmgr (createUFS). Although no real dynamic block creation can be achieved through this approach, the perception of this is given.

The representation as object references as PIds is also different than the CORBA world. In CORBA, object references contain sufficient information so that a client can decide whether they wish to invoked that service or not - at

least from a syntactic point of view! The same is not true for SDL. Possession of a PId by another process does not allow that process to see what signals can be sent to the process instance referenced by that PId1.

There are also other minor problems due to the differences in keywords from the ODL, IDL and SDL worlds, e.g. start is a valid IDL operation name but not a valid SDL procedure name. These issues were easily resolved in TOSCA since the frameworks were created in parallel, e.g. names such as ufsstart were found that were satisfactory to all three languages.

5 SDL Framework Development

Developing a framework so that it removes large parts of the problem of service design, thus expediting the creation process, whilst still offering a means to create numerous different kinds of services is an especially challenging activity. To produce successful frameworks requires that the points where design decisions are made are flexibility points. Using frameworks to produce services then requires that these flexibility points are made available so that new design choices can be taken to produce new services. Perhaps the hardest part of the framework development process is the identification of these flexibility points [9].

In TOSCA we focused on a small set of flexibility points. This set of flexibility points allowed us to produce a multitude of different services with different types of behaviour. Specifically, we chose the following flexibility points:

- start up, suspension, resumption and termination of *user* and *service* sessions.

In producing a framework it is necessary to have fixed places where the flexibility points are to exist. Thus it is necessary to represent the points of flexibility directly in the design of the framework, but the actual behaviour associated with these flexibility points is effectively NULL until they are specialised. To achieve this we introduced appropriate IDL operations that were associated with the appropriate objects in the framework design. An example of the kind of IDL associated with the UFSmgr described earlier is:

```
interface i_UFSmgr : i_CO_lifecycle {
  void suspendSessionRequest();            // suspend a users's session
  void terminateSessionRequest();          // terminate a user's session
  void suspendAll();        // suspend USM and all managed objects
  void requestObject(inout NamedObject obj);    // create handlers
  oneway void ufsstart(); // not implemented in framework - specialised!
```

```
  oneway void ufssuspend();              // dto.
  oneway void ufsresume();               // dto.
  oneway void ufsstop();                 // dto.
                          //..... other operations
};
```

We point out that the behaviour with the other IDL operations can be implemented directly, i.e. before specialisation. As with other IDL language mappings, client stubs and server skeletons are generated from Y.SCE. These act as templates whose behaviour is to be filled in through inheritance. These stubs and skeletons are placed in two SDL packages (Name_Interface and Name_Definition). The Name_Interface package contains the interface specifications in the form of data types, signals, remote procedures, signallists etc. Figure 5 gives an example of the kind of SDL generated focusing on the i_UFSmgr interface of the UFSmgr object:

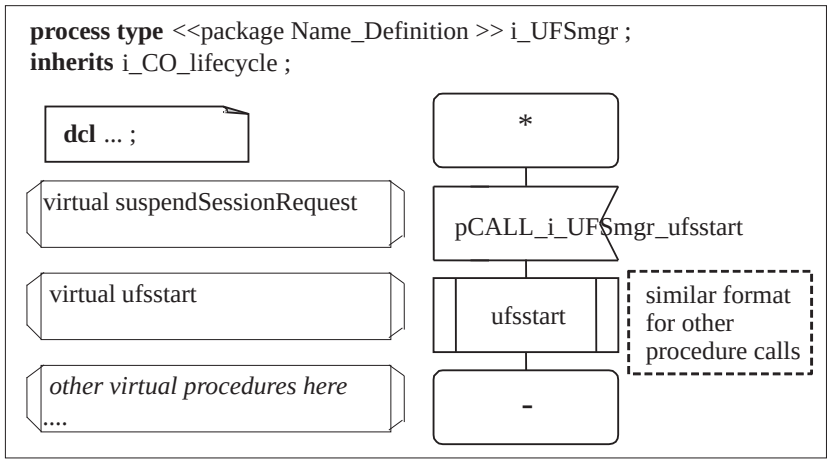

Fig. 5. Example of the Contents of the Name_Definition Package

The virtual procedure for the ufsstart (and all oneway operations) consist of a virtual start transition followed by an immediate exit. In non-oneway operations, the generated procedures contain a pREPLY_ signal of the appropriate kind. Along with the virtual procedure definitions, signals and (asterisk) states are also generated that result in the procedures being called.

As an example of the way in which the generated SDL server skeletons can have their core behaviour inserted, i.e. the behaviour before they are specialised, we consider the implementation of the i_UFSmgr interface (i_UFSmgrImp) of the UFS object given previously. The default behaviour for the UFSmgr is that it creates a control window handler only. A simplified example of the structure of this object is given in figure 6.

This process type is parameterised with (amongst other things) the reference to the user application, i.e. the PId for the ssUAP manager process. When an

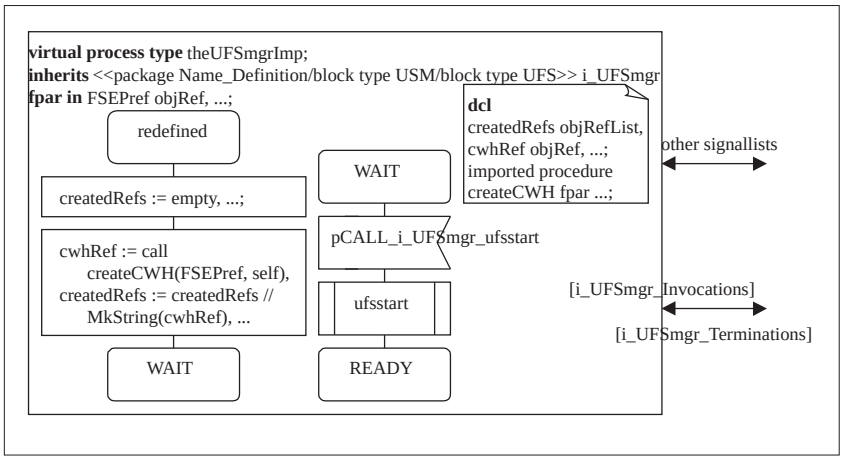

Fig. 6. Structure of Basic UFSmgr

instance of this process type is created, initialisation of local variables is done, e.g. the list of created references is set to empty, and the default behaviour of creating a control window handler is made. As discussed, this requires that the necessary exported remote procedure is imported. Following this default behaviour, the UFSmgr is ready to be specialised, i.e. it is in a state where it can accept the signal pCALL_i_UFSmgr_ufsstart.

5.1 Advantages of SDL for Framework Development

SDL has many advantages when used to develop frameworks [17]. Many of these advantages stem directly from its support of object-orientation. For example, the ability to directly re-use (through redefinition) the Y.SCE generated SDL stubs and skeletons contained in the Name_Interface and Named_Definition packages allows development of specifications to be made in a constructive (and CORBA-like) manner.

The representation of flexibility points is also easily achieved in SDL through procedures which can be called but have null behaviours, i.e. start and exit. This allows for the behaviour of the framework as a whole to be checked without necessarily having any specialisation taking place, e.g. the basic USM behaviour (and SSM and SF) behaviours can be checked to ensure the framework as a whole correctly represents the informal (textual) requirements.

This representation of holes also means that the SDL frameworks can be used to derive test cases. These can then be used to check that SDL based specialisations of the frameworks are valid, i.e. SDL service models, and that the service implementations derived from the C++/CORBA frameworks are valid.

The SDL language itself supports the development of frameworks, however, in TOSCA problems arose in the usage of tools when developing frameworks. This was apparent when the framework design changed. Ideally, when the ODL or IDL for the framework changed, the existing packages upon which the SDL framework was based, could be replaced with these new packages. This process should not have impinged upon the behaviour specified in the SDL framework that was independent of the new design change. Unfortunately, this process was not possible. When new packages were generated, file names were not guaranteed to be distinct from those existing in the SDL framework, i.e. not just the existing packages. This could (and did) lead to situations where the behaviour specified in the framework was lost due to a new file being generated that had the same name as an existing framework file, i.e. a file that had SDL behaviour inserted was overwritten by a new skeleton. To address this issue, the SDL stubs were manually edited. Clearly, this is unsatisfactory, and requires further consideration.

As described earlier, in TOSCA we developed in parallel a C++ implementation and a SDL model of the same framework. In the C++ and Orbix world there are two signal queues connected to each object. That is, incoming requests to that object are collected in one queue, while a second queue accepts replies to earlier requests. Hence an object cannot deadlock or drop an incoming request (through implicit signal consumption) because it received an incoming request at a time it expected a reply. Although there are solutions such as using the save construct, or processes to represent queues, these are not straightforward to implement and somewhat inelegant.

Opposed to C++, a missing aspect of the object-orientation of SDL is that a user cannot be forced to specialise certain framework parts. That is, a component declared as virtual can be specialised in the service model but the specification would also be semantically correct if it is not.

6 Service Creation – Specialising the Framework

As an example specialisation of the framework we show how a videophone service can be created. This services supports two user roles: Caller and Callee. There may only be one instance of these in the service at a time. The caller and callee both have windows on their user application (ssUAP) which can be used for terminating or suspending their respective participations in the service. The caller and callee differ in that the caller also has an invitation window (for inviting the callee) and the callee is terminated from the session

after thirty seconds of suspension. That is, the callee should resume within thirty seconds or they are automatically quit.

The user applications (ssUAP) associated with users are not modelled in the framework. Instead the real C++/CORBA implementation of these objects are used, that is they are driven by a simulation of the SDL model. The objects themselves allow for the dynamic manipulation of the user interface, e.g. new windows or buttons can be added. The signals to achieve this come from the SDL system. Specifically, from the specialisable procedure ufsstart. The objects that deal with user application requests in the USM all support a callback interface (process type). It is instances of these process types that the events raised by the user, e.g. through pushing buttons on their ssUAP, are sent. Thus in the videophone example, the specialised ufstart for the caller should create an invitation window handler. This handler then requests the user application to add the appropriate window and the callback references are established. We note that this is the most simple scenario since the invitation window handler is a predefined component in the framework. Other more complex can be achieved however. Examples of how this is achieved are given in [9]. The specialised procedure for the caller ufsstart is shown in figure 7.

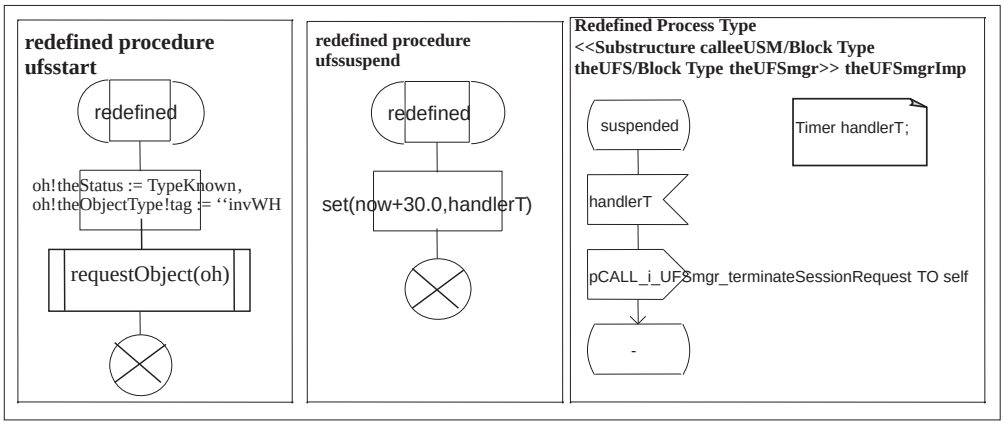

Fig. 7. Specialising SDL for Caller and Callee in Videophone Service

For the callee, procedure ufssuspend must be specialised. To achieve this an SDL timer is introduced and set to time (now + 30). The signal which is generated by the timer is received in the UFSmgr and results in sending a signal to terminate the callee. In the specialisable procedure ufsresume (called during resumption of a user), the timer is reset. Thus if ufsresume is executed before the timer signal is consumed (less then 30 seconds) the timer is stopped and the user can resume into the session.

Using the framework for producing services requires that the framework is saved as a package. This implies minor modifications, like the removal of the upper most block instances with their connecting channels. To specialise the framework to create a service, the SDL package representing the framework was used. Both simple and virtual inheritance were used to specialise the components in the framework. Simple inheritance was used at the top most block level, e.g. the USM block level. Subsequent block types, e.g. the UFS block type as well as process types and procedures were reused by virtual inheritance. This was necessary since virtual inheritance allows for the communication links, i.e. channels and signalroutes in the framework to be reused (and possibly extended). Keeping the channels and signalroutes in the specialisation was essential for being able to model frameworks in SDL - otherwise the 'added-value' of frameworks in storing the design and communication links, is lost.

On the other hand, virtual inheritance does not allow for multiple redefinitions in one scope. However, considering the classes of services targeted in TOSCA, this is a crucial property to have. Different types of users in a service are represented by different types of USMs, or in other words, different specialisations of the base USM in the framework. This is used for modelling the various attributes of different user types, e.g. a chairman and participants in a conference service. As a result, it is not possible to use virtual inheritance for the top-level block types: simple inheritance is used instead. This implies that since the communication paths between blocks at the top most level are not part of the framework they need to be generated by the paradigm tool.

6.2 Problems with using SDL for Framework Specialisation

A difficulty encountered in the development of the paradigm tool was the fact that process communication (via signals) always requires an explicitly drawn signal path between the communicating processes. This is also true if signals are directly sent to a specific process identifier (PId) of a process in the system. We note that this is not strictly true according to the SDL standard [21], however, it was true with the SDT tool.

For TOSCA, this meant a further deviation from the C++/CORBA framework implementation and how it is specialised. In the CORBA world, possession of an interface reference is sufficient to be able to communicate with the interface instance - assuming of course that the reference is still valid, i.e. it references an existing and available interface. Furthermore, as far as automatic

SDL generation by the paradigm tool is concerned, it is not sufficient to only know the sender and potential receiver of a signal but corresponding channels and signalroutes need to be generated as well. As the necessary paths often involve multiple hierarchies of block and process types, the generation of the channels and signalroutes cannot be achieved in a straightforward manner.

A further difference between the CORBA world and the SDL world, is that clients in possession of an interface reference can see whether they wish to invoke the service represented by the interface reference. In SDL this is not the case. One way that this might be resolved is through the development of interface repositories where possession of an interface reference (PId) enables clients to query the functionality offered by that PId, e.g. the signals or remote procedures that can be invoked and the associated terminations. Ideally, this facility should be part of the ODL/IDL to SDL mapping process.

7 Deriving Test Cases from the SDL Models

As stated in section 2, the framework and services derived from the framework are used to generate test cases. In the former case, these test cases are used to check the minimum conformance requirement of the SDL and C++/CORBA services derived from the frameworks. That is, these framework test cases are used to ensure that the created services are valid specialisations of the associated frameworks. In the latter case, the test cases generated are used to check the conformance relation between the SDL and C++/CORBA service models.

Several tools exist within the Telelogic TAU toolset that allow for the derivation of tests from SDL specifications. The Autolink tool of the SDT Validator allows for the semi-automatic generation of TTCN test suites based on SDL specifications. Development of test suites from the SDL models can also be made interactively through the SDT TTCN link tool. This tool provides an environment that links the SDL specification world represented by the Specification Design Tool (SDT) with the testing world represented by the ITEX tool. Once a TTCN link executable is generated from the specification it may be opened with ITEX and used to generate the declarations used to test the system. In effect this corresponds to generating mappings for the SDL channel names, the signals they carry and the parameters associated with these signals that the specification has with its environment. The SDL channels are mapped to points of control and observation (PCO) type declarations, the signals are mapped to ASN.1 abstract service primitive (ASP) type definitions and the signal parameters are mapped to ASN.1 type definitions. An extra TTCN table is also generated called OtherwiseFail. This table is used to catch all other ASPs at the PCOs, i.e. signals on channels, other than those listed

in the test case through an ?OTHERWISE statement. These result in a fail verdict for the test. This table also accepts arbitrary timeout signals which result in an inconclusive test through a ?TIMEOUT statement. This table is used as a default case for the test suite.

Having generated the static parts of the tests, the dynamic parts and the constraint parts associated with the test case can be developed through synchronising the TTCN test case with the SDL system. Once synchronised, the messages to be sent and received can be selected, i.e. the PCOs used (channels to/from the specification) together with the ASN.1 ASPs they carry from the list of possible SDL signals at that time. Once a PCO and ASN.1 ASP has been selected the constraints associated with the signal, e.g. the values of the parameters being sent or the acceptable values that are being received, can be set and a verdict be assigned to the test case.

7.1 Problems with Test Case Generation and Execution

The generation of test cases from the SDL model for the framework and services derived from the framework is not without problems. Many of these stem from the necessary complexity of the system being specified. As stated in section 4.2, distributed systems are complex and typically consist of many objects interacting in non-trivial ways, i.e. they pass (and accept) complex data structures when interacting. The result of this is that SDL models of such systems are themselves complex due to the similar level of abstraction upon which they are based, i.e. IDL.

The more complex a specification is, the less easy it is to check through tools. Checks that are made during test case generation are typically based on exploring the state space of the specification. However, it was found within TOSCA that this activity was not well supported due to the specification complexity. A typical manifestation of the problem was a deadlock of the tools when trying to generate test cases.

A further problem with test case generation was the lack of support for PIds in TTCN or ASN.1. In distributed systems, object references, i.e. PIds, play a central role in providing location transparency. These are passed around as parameters in many processes, and also accepted by the environment. For example, when a service is first started references to the ssUAP manager (FSEPmgr) are passed in as a parameter. These are then used by the specification when adding buttons etc to the ssUAP, e.g. signals are directed to the FSEPmgr reference and not just to the environment. TTCN does not support a mapping for PIds however. Hence it was not possible to generate test cases for the specification directly. Instead, the specification had to be modified so

that it never accepted or produced PIds. Rather, it accepted another data type (string) and this was then mapped (within the specification) to the appropriate PId. Thus the specification required look-up tables to be specified from PIds and strings (and vice-versa) for testing.

The abstract test cases generated were to be executed through a TTCN/CORBA gateway against the CORBA based service implementations [26]. Here again, however, the test cases generated through the Telelogic testing tools could not be executed directly against the service implementation. The TTCN/CORBA gateway has a different representation of PCOs. This tool maps PCOs to object references. Test suite operations are given that allow the object references of the service implementation to be accessed, e.g. the stringified form of these references can be read in from a file, and subsequently used in testing. Effectively, this issue stems from the TAU tools treating service model specifications as black boxes when deriving test cases, when tests of component interfaces inside of the specification model are really what is needed. Put another way, SDL channels do not have an equivalent counterpart in the distributed services considered in TOSCA. That is, these services are tested through interacting at their interfaces, i.e. the external interfaces of the objects they are composed of. Channels do not correspond to interfaces in the TOSCA service models. The same channel can be (and is) used to send and receive many signals from many different objects within a single block. This is further exacerbated by having multiple instances of the same block type, e.g. where there might be twenty observers in a service. It is of course possible to have twenty block instances with separate channels connected to the environment, however, this approach is neither practical nor scalable. For example, it would require that twenty identical specialisations of the framework USM were made. To resolve this, the test cases were manually edited so that the PCOs generated by the TAU tools were replaced by CORBA interface references obtained via TTCN/CORBA gateway based test suite operations.

8 Conclusions

This paper has outlined the work in the ACTS TOSCA project. A framework and paradigm tool based approach has been chosen to create telecommunications services. The overall goal of the project is to create multimedia based services quickly and of high quality. Starting with ODL/IDL specifications and an informal behaviour description of components based on the TINA service session, C++/CORBA and SDL based frameworks were developed in parallel. These two frameworks were then used to create service implementations and service models respectively.

We have highlighted some of the advantages and disadvantages of applying SDL and its associated tools for this purpose. On the plus side, is that SDL allows ODL/IDL based models to be specified at all. It is one of the few languages to address current techniques and approaches to (distributed) software development. The language also has many features that make it apposite for framework development, e.g. the straightforward representation of flexibility points means that SDL frameworks can be simulated and tested even though they are incomplete models of software. The usage of frameworks is a natural feature of SDL due to its support for object-orientation and package constructs. An added bonus is that tools are available that enable test cases to be semi-automatically generated.

The picture is not all rosy however. One of the main advantage of having SDL models of systems is through the tools that can be used to investigate the behaviour of those systems. Unfortunately, due to the complexity of the services in TOSCA, usage of tools that enable such detailed behaviour investigations were limited due mainly to the well known problem of state space explosion. This problem was also manifest when generating test cases from the SDL service specifications. Other issues also arose when test case generation was attempted however. For example, it should not necessarily be the case that testing of distributed systems is based on black box testing, i.e. where channels are given as PCOs. Instead, testing of individual components within the service should be possible - ideally, the interfaces (processes) that are to be tested.

Many of the issues that arose in this work were due to the different mappings stemming from the ODL/IDL to SDL and CORBA IDL to C++ worlds. In particular this was caused by the SDL approach attempting to resolve issues that the CORBA world has yet to address, e.g. multiple interface objects. It is hoped and expected that many of the issues in this paper will be resolved through the upcoming SDL2000 standard, e.g. dynamic block (multiple interface object) creation. However, many of the other issues discussed here are based on the need for alignment and extensions to existing SDL tools. In particular, those tailored to distributed service creation.

Acknowledgements

The authors are indebted to the partners in the TOSCA project. The TOSCA consortium consists of Teltec DCU, Silicon & Software Systems Ltd, British Telecommunications, University of Strathclyde, Centro Studi e Laboratori di Telecommunicazioni SpA, Telelogic, Lund Institute of Technology, GMD and Ericsson.

References

[1] M. Fayad and D. Schmidt, *Object-Oriented Application Frameworks*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 32-38.

[2] R. Johnson, *Frameworks = (Components + Patterns)*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 39-42.

[3] H. Schmid, *Systematic Framework Design*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 48-51.

[4] D. Bäumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle and H. Züllighoven, *Framework Development for large Systems*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 52-59.

[5] S. Demeyerm T. Meijler, O. Nierstrasz and P. Steyaert, *Design Guidlines for 'Tailorable' Frameworks*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 60-64.

[6] D. Brugali, G. Menga and A. Aarsten, *The Framework Lifespan*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 65-68.

[7] W. Codenie, K. de Hondt, P. Steyaert and A. Vercammen, *From Custom Applications to Domain-Specific Frameworks*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 71-77.

[8] D. Schmidt and M. Fayad, *Lessons Learned Building Reusable OO Frameworks for Distributed Software*, Communications of the ACM, Volume 40, Number 10, October 1997, pp. 85-87.

[9] R. Sinnott and M. Kolberg, *Business-Oriented Development of Telecommunication Services with SDL*, Proceedings of OOPSLA-98 Workshop on Precise Behaviour Specifications of Object-oriented Systems and Business Specifications, Vancouver, Canada, October 1998.

[10] TOSCA Consortium Deliverable 9, *Final Specification of Process Architecture*, http://www.teltec.dcu.ie/tosca

[11] TOSCA Consortium Deliverable 11, *User Trial Report on Embedded Methods and Tools*, http://www.teltec.dcu.ie/tosca

[12] R. Sinnott and M.Kolberg, *Engineering Telecommunication Services With SDL*, in Formal Methods for Open, Object-Based Distributed Systems, P. Ciancarini, A. Fantechi and R. Gorriere (Eds.), Kluwer Academic Publishers, 1999, pp. 187-203.

[13] M. Kolberg and E. Magill, *Service and Feature Interactions in TINA*, in Feature Interactions in Telecommunications and Software Systems V, K. Kimbler and L.G. Bouma (Eds.), IOS Press, 1998, pp.78-84.

[14] M. Kolberg, R.O. Sinnott and E. H. Magill, *Engineering of Interworking TINA-based Telecommunication Services*, Proceedings of Telecommunications Information Networking Architecture Conference, Oahu, Hawaii, April 1999.

[15] A. Olsen, D. Demany, E. Cardoso, F. Lodge, M. Kolberg, M. Björkander and R. Sinnott, *The Pros and Cons of Using SDL for Creation of Distributed Services*, in Intelligence in Services and Networks - Paving the Way for an Open Service Market, H. Zuidweg, M. Campolargo, J. Delgado, A. Mullery (Eds.), pp. 342-354, Lecture Notes in Computer Science, Vol. 1597, Springer Verlag, 1999.

[16] *The Common Object Request Broker Architecture and Specification: Revision 2.0*, Object Management Group, Inc., Framingham MA., July 1995.

[17] R. Sinnott, *Frameworks: The Future of Formal Software Development*, Journal of Computer Standards and Interfaces, special edition on Semantics of Specifications, August 1998.

[18] R. Sinnott and M. Kolberg, *Creating Telecommunication Services based on Object-Oriented Frameworks and SDL*, Proceedings of Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'99), Saint Malo, France, 1999.

[19] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modelling and Design*, Prentice Hall 1991.

[20] R. Johnson and V. Russo, *Reusing Object-Oriented Designs*, Urbana, Ill., May 1991.

[21] International Consultative Committee on Telegraphy and Telephony, *SDL - Specification and Description Language, CCITT Z.100*, International Telecommunications Union, Geneva, Switzerland, 1992.

[22] For more information see http://www.fokus.gmd.de/minos/y.sce.

[23] TINA-C, *TINA Object Definition Language Manual*, Version 2.3, July 1996.

[24] Telelogic AB, *Getting Started Part 1 - Tutorials on SDT Tools*, Telelogic AB, 1997.

[25] I. Schieferdecker, M. Li, A. Hoffmann, *Conformance Testing of TINA Service Components - the TTCN/CORBA Gateway*, Proceedings of the Intelligence in Networks and Services Conference 1998, Antwerp, May 1998.

[26] M. Li, *Testing Computational Interfaces of TINA Services using TTCN and CORBA*, Diplomarbeit, Department of Electrical Engineering, Telecommunication Network Group, Technical University Berlin, 1997.

[27] TINA-C, *Service Architecture*, Version 5.0, 16 June 1997.

[28] M. Born, A. Hoffmann, M. Winkler, J. Fischer, N. Fischbeck, *Towards a Behavioural Description of ODL*, Proceedings of TINA 97 Conference, Chile.

[29] M. Björkander, *Mapping IDL to SDL*, Telelogic AB, 1997.

[30] *Basic Reference Model of ODP - Part 2: Foundations*, ISO/IEC International Standard 10746-2, ITU-T Recommendation X.902, Geneva, Switzerland 1997.